autogluon_each_location

October 18, 2023

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*30
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

use_groups = False
n_groups = 8

auto_stack = False
num_stack_levels = 0
num_bag_folds = 8
num_bag_sets = 20

use_tune_data = True
use_test_data = False
tune_and_test_length = 0.25 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False
sample_weight_estimated = 1

run_analysis = False
```

[2]:
```
import pandas as pd
import numpy as np
```

```python
import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
       'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
       'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    X_shifted = X[X.index.minute==0][columns].copy()
    # loop through all rows and check if index + 1 hour is in the index, if so␣
↪get that value, else nan
    count1 = 0
    count2 = 0
    for i in range(len(X_shifted)):
        if X_shifted.index[i] + pd.Timedelta('1 hour') in X.index:
            count1 += 1
            X_shifted.iloc[i] = X.loc[X_shifted.index[i] + pd.Timedelta('1␣
↪hour')][columns]
        else:
            count2 += 1
            X_shifted.iloc[i] = np.nan

    print("COUNT1", count1)
    print("COUNT2", count2)

    X_old_unshifted = X[X.index.minute==0][columns]
    # rename X_old_unshifted columns to have _not_shifted at the end
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
↪columns]

    # put the shifted columns back into the original dataframe
    #X[columns] = X_shifted[columns]



    date_calc = None
    if "date_calc" in X.columns:
```

```python
        date_calc = X[X.index.minute == 0]['date_calc']

    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
```

```
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
 ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
 ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =␣
 ↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
 ↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)
```

```python
    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
 ↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
```

```
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
```

1 Feature enginering

[3]:
```
import numpy as np
import pandas as pd

X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
 ↪inplace=True)
```

```
for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

print(X_train.head())




if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())




to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
 ↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
 ↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
 ↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
 ↪gm3", "air_density_2m:kgm3"]
```

```python
X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00                     7.700              1.22825
2019-06-02 23:00:00                     7.700              1.22350
2019-06-03 00:00:00                     7.875              1.21975
2019-06-03 01:00:00                     8.425              1.21800
2019-06-03 02:00:00                     8.950              1.21800

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00           1728.949951               0.000000
2019-06-02 23:00:00           1689.824951               0.000000
2019-06-03 00:00:00           1563.224976               0.000000
2019-06-03 01:00:00           1283.425049            6546.899902
2019-06-03 02:00:00           1003.500000          102225.898438

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00             0.00       1728.949951              0.0
2019-06-02 23:00:00             0.00       1689.824951              0.0
2019-06-03 00:00:00             0.00       1563.224976              0.0
2019-06-03 01:00:00             0.75       1283.425049              0.0
2019-06-03 02:00:00            23.10       1003.500000              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
2019-06-02 22:00:00      280.299988          0.000          0.000000  …
2019-06-02 23:00:00      280.299988          0.000          0.000000  …
2019-06-03 00:00:00      280.649994          0.000          0.000000  …
2019-06-03 01:00:00      281.674988          0.300       7743.299805  …
2019-06-03 02:00:00      282.500000         11.975      60137.601562  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 22:00:00   286.225006           100.000000  40386.476562
2019-06-02 23:00:00   286.899994           100.000000  33770.648438
2019-06-03 00:00:00   286.950012           100.000000  13595.500000
2019-06-03 01:00:00   286.750000           100.000000   2321.850098
2019-06-03 02:00:00   286.450012            99.224998  11634.799805

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
```

```
ds
2019-06-02 22:00:00             3.600              -3.575
2019-06-02 23:00:00             3.350              -3.350
2019-06-03 00:00:00             3.050              -2.950
2019-06-03 01:00:00             2.725              -2.600
2019-06-03 02:00:00             2.550              -2.350

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 22:00:00               -0.500                      0.0
2019-06-02 23:00:00                0.275                      0.0
2019-06-03 00:00:00                0.750                      0.0
2019-06-03 01:00:00                0.875                      0.0
2019-06-03 02:00:00                0.925                      0.0

                     is_estimated      y  location
ds
2019-06-02 22:00:00             0   0.00         A
2019-06-02 23:00:00             0   0.00         A
2019-06-03 00:00:00             0   0.00         A
2019-06-03 01:00:00             0   0.00         A
2019-06-03 02:00:00             0  19.36         A

[5 rows x 48 columns]
```

[4]:
```python
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
 ↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
import numpy as np

def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
 ↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output␣
 ↪dataset.
```

```
    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[5]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
```

```
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
test_set = TabularDataset(data[data["ds"] >= split_time])

# shuffle test_set and only grab tune_and_test_length percent of it, rest goes␣
↪to train_set
test_set, new_train_set = split_and_shuffle_data(test_set, 40,␣
↪tune_and_test_length)


print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))
print("Length of test set", len(test_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = None
if use_tune_data:
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            # randomly shuffle the loc_test_set
```

```
            loc_tuning_data, loc_test_data =␣
↪split_and_shuffle_data(loc_test_set, 40, 0.5)
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
↪shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in␣
↪the tuning data.
    if weight_evaluation:
        tuning_data = normalize_sample_weights_per_location(tuning_data)


else:
    if use_test_data:
        test_data = test_set
        print("Shape of test", test_data.shape[0])


train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
if use_tune_data:
    tuning_data = TabularDataset(tuning_data)
if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Length of train set before adding test set 82026
Length of train set after adding test set 87486
Length of test set 5459
Shape of tuning 5459
```

[6]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
 ↪label="y", sample=None)
```

[7]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

2 Starting

[8]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
 ↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 96
Now creating submission number: 97
New filename: submission_97
```

[9]:
```
predictors = [None, None, None]
```

[10]:
```
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
 ↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
 ↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
 ↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
 ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
```

```
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
↪drop(columns=["ds"]),
        time_limit=time_limit,
        # presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_97_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
```

```
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   200.38 GB / 315.93 GB (63.4%)
Train Data Rows:    31872
Train Data Columns: 32
Tuning Data Rows:    2187
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 649.68162,
1178.37671)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132000.72 MB
        Train Data (Original)  Memory Usage: 10.42 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…

Training model for location A…

        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
```

```
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 7.94 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 1799.84s of
the 1799.84s of remaining time.
        -132.836         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.38s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 1799.33s of
the 1799.33s of remaining time.
        -132.5631        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
```

```
        0.39s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1798.84s of the
1798.84s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -93.681  = Validation score   (-mean_absolute_error)
        28.46s   = Training   runtime
        18.91s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1759.66s of the
1759.66s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -97.2854         = Validation score   (-mean_absolute_error)
        23.71s   = Training   runtime
        5.64s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 1731.84s of
the 1731.84s of remaining time.
        -105.3243        = Validation score   (-mean_absolute_error)
        7.75s    = Training   runtime
        1.13s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1721.7s of the
1721.7s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -103.3471        = Validation score   (-mean_absolute_error)
        192.01s  = Training   runtime
        0.09s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 1528.51s of the
1528.51s of remaining time.
        -107.644         = Validation score   (-mean_absolute_error)
        1.6s     = Training   runtime
        1.14s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1524.54s of
the 1524.54s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -108.0147        = Validation score   (-mean_absolute_error)
        38.33s   = Training   runtime
        0.48s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 1484.4s of the
1484.4s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -104.4468        = Validation score   (-mean_absolute_error)
        5.82s    = Training   runtime
        0.3s     = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 1476.48s of
the 1476.47s of remaining time.
```

```
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -95.8917         = Validation score   (-mean_absolute_error)
        125.34s  = Training   runtime
        0.31s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 1349.69s of the
1349.69s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -96.3103         = Validation score   (-mean_absolute_error)
        92.93s   = Training   runtime
        16.21s   = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1247.25s of the
1247.25s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -93.7227         = Validation score   (-mean_absolute_error)
        58.55s   = Training   runtime
        38.01s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1209.85s of the
1209.85s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -97.5177         = Validation score   (-mean_absolute_error)
        48.72s   = Training   runtime
        10.54s   = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1179.89s of the
1179.89s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -103.1179        = Validation score   (-mean_absolute_error)
        387.72s  = Training   runtime
        0.2s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 982.81s of
the 982.8s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -108.2172        = Validation score   (-mean_absolute_error)
        76.99s   = Training   runtime
        0.94s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 941.47s of the
941.46s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -103.2888        = Validation score   (-mean_absolute_error)
        13.9s    = Training   runtime
        0.66s    = Validation runtime
```

```
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 931.77s of the
931.77s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -94.7112         = Validation score   (-mean_absolute_error)
        240.82s  = Training   runtime
        0.61s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 814.66s of the
814.66s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -95.9885         = Validation score   (-mean_absolute_error)
        183.8s   = Training   runtime
        36.7s    = Validation runtime
Completed 2/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
708.54s of remaining time.
        -89.9806         = Validation score   (-mean_absolute_error)
        0.45s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 1091.93s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_97_A/")
```

[11]:
```python
import matplotlib.pyplot as plt

leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_test_data:
        lb = predictors[i].leaderboard(test_data[test_data["location"] == loc])
        lb["location"] = loc
        plt.scatter(test_data[test_data["location"] == loc]["y"].index,
 ↪test_data[test_data["location"] == loc]["y"])
        if use_tune_data:
            plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,
 ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.show()

        return lb
    else:
        return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```

[12]:
```python
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
```

```
leaderboards[1] = leaderboard_for_location(1, loc)
```

```
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_97_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   196.72 GB / 315.93 GB (62.3%)
Train Data Rows:    31020
Train Data Columns: 32
Tuning Data Rows:    1797
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 99.56591, 196.469)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130075.61 MB
        Train Data (Original)  Memory Usage: 10.04 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
```

```
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 7.65 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}

Training model for location B…

Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 1799.84s of
the 1799.84s of remaining time.
        -26.8714         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
```

```
        0.38s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 1799.19s of
the 1799.19s of remaining time.
        -26.8453         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.39s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1798.71s of the
1798.7s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -17.086  = Validation score   (-mean_absolute_error)
        29.05s   = Training   runtime
        18.94s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1763.74s of the
1763.73s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -16.5481         = Validation score   (-mean_absolute_error)
        32.3s    = Training   runtime
        13.98s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 1726.94s of
the 1726.94s of remaining time.
        -17.8185         = Validation score   (-mean_absolute_error)
        8.83s    = Training   runtime
        1.12s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1715.93s of the
1715.93s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -17.4238         = Validation score   (-mean_absolute_error)
        194.87s  = Training   runtime
        0.1s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 1519.75s of the
1519.75s of remaining time.
        -17.1018         = Validation score   (-mean_absolute_error)
        1.54s    = Training   runtime
        1.13s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1515.93s of
the 1515.93s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.344  = Validation score   (-mean_absolute_error)
        39.37s   = Training   runtime
        0.47s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 1474.64s of the
1474.64s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
```

```
        -16.6706         = Validation score   (-mean_absolute_error)
        82.64s   = Training   runtime
        19.6s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 1386.19s of
the 1386.19s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.0656         = Validation score   (-mean_absolute_error)
        185.86s  = Training   runtime
        0.31s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 1198.95s of the
1198.95s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.4634         = Validation score   (-mean_absolute_error)
        95.34s   = Training   runtime
        19.27s   = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1093.48s of the
1093.48s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -17.0788         = Validation score   (-mean_absolute_error)
        58.56s   = Training   runtime
        34.22s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1057.11s of the
1057.11s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -16.5591         = Validation score   (-mean_absolute_error)
        63.31s   = Training   runtime
        29.79s   = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1019.24s of the
1019.24s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -17.3786         = Validation score   (-mean_absolute_error)
        390.21s  = Training   runtime
        0.2s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 822.6s of the
822.6s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.142  = Validation score   (-mean_absolute_error)
        77.13s   = Training   runtime
        0.98s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 782.26s of the
782.26s of remaining time.
```

```
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -16.5324         = Validation score   (-mean_absolute_error)
        164.4s    = Training   runtime
        41.45s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 691.33s of the
691.33s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.7927         = Validation score   (-mean_absolute_error)
        349.19s   = Training   runtime
        0.65s     = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 526.39s of the
526.39s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.4482         = Validation score   (-mean_absolute_error)
        189.08s   = Training   runtime
        43.7s     = Validation runtime
Completed 2/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
415.81s of remaining time.
        -12.7634         = Validation score   (-mean_absolute_error)
        0.47s     = Training   runtime
        0.0s      = Validation runtime
AutoGluon training complete, total runtime = 1384.68s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_97_B/")
```

[13]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_97_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   192.23 GB / 315.93 GB (60.8%)
Train Data Rows:    24594
Train Data Columns: 32
Tuning Data Rows:    1475
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
```

```
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, -0.0, 79.8926, 168.407)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    129625.83 MB
        Train Data (Original)  Memory Usage: 7.98 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 6.07 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.13s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
```

```
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 1799.87s of
the 1799.86s of remaining time.

Training model for location C…

        -23.875  = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.26s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 1799.5s of the
1799.5s of remaining time.
        -23.8095         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.27s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1799.15s of the
1799.15s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.2146         = Validation score   (-mean_absolute_error)
        25.99s   = Training   runtime
        14.98s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1767.79s of the
1767.78s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

```
ParallelLocalFoldFittingStrategy
        -13.3767         = Validation score   (-mean_absolute_error)
        24.19s   = Training   runtime
        5.12s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 1739.72s of
the 1739.71s of remaining time.
        -16.5325         = Validation score   (-mean_absolute_error)
        4.66s    = Training   runtime
        0.75s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1733.73s of the
1733.72s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.3629         = Validation score   (-mean_absolute_error)
        185.33s  = Training   runtime
        0.09s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 1547.16s of the
1547.15s of remaining time.
        -16.3493         = Validation score   (-mean_absolute_error)
        0.98s    = Training   runtime
        0.76s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1544.76s of
the 1544.76s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.5328         = Validation score   (-mean_absolute_error)
        30.52s   = Training   runtime
        0.44s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 1512.48s of the
1512.48s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.5911         = Validation score   (-mean_absolute_error)
        58.44s   = Training   runtime
        4.92s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 1450.02s of
the 1450.02s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.7827         = Validation score   (-mean_absolute_error)
        86.64s   = Training   runtime
        0.31s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 1362.02s of the
1362.01s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.8691         = Validation score   (-mean_absolute_error)
        85.79s   = Training   runtime
```

```
        10.79s   = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1265.29s of the
1265.29s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.2878         = Validation score   (-mean_absolute_error)
        52.18s   = Training   runtime
        28.82s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1231.3s of the
1231.3s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.2583         = Validation score   (-mean_absolute_error)
        48.46s   = Training   runtime
        9.68s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1201.56s of the
1201.56s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.326  = Validation score   (-mean_absolute_error)
        371.29s  = Training   runtime
        0.17s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1014.3s of
the 1014.3s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.5944         = Validation score   (-mean_absolute_error)
        61.15s   = Training   runtime
        0.84s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 981.43s of the
981.43s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.6625         = Validation score   (-mean_absolute_error)
        101.13s  = Training   runtime
        7.07s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 933.99s of the
933.99s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.8264         = Validation score   (-mean_absolute_error)
        168.16s  = Training   runtime
        0.66s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 850.89s of the
850.89s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
```

```
        -12.685  = Validation score   (-mean_absolute_error)
        170.66s  = Training   runtime
        22.33s   = Validation runtime
Repeating k-fold bagging: 3/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 751.89s of the
751.89s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.2931         = Validation score   (-mean_absolute_error)
        78.29s   = Training   runtime
        44.19s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 716.6s of the 716.6s
of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.2496         = Validation score   (-mean_absolute_error)
        71.61s   = Training   runtime
        14.21s   = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 687.35s of the
687.35s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.3702         = Validation score   (-mean_absolute_error)
        557.27s  = Training   runtime
        0.25s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 500.02s of
the 500.02s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.5832         = Validation score   (-mean_absolute_error)
        91.9s    = Training   runtime
        1.25s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 466.39s of the
466.39s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.7012         = Validation score   (-mean_absolute_error)
        142.29s  = Training   runtime
        8.29s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 420.08s of the
420.08s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.7879         = Validation score   (-mean_absolute_error)
        252.17s  = Training   runtime
        0.96s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 334.23s of the
334.23s of remaining time.
```

```
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.6813         = Validation score   (-mean_absolute_error)
        259.25s  = Training   runtime
        35.99s   = Validation runtime
Completed 3/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
226.04s of remaining time.
        -11.8138         = Validation score   (-mean_absolute_error)
        0.42s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 1574.41s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_97_C/")
```

[14]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

3 Submit

[15]:
```
import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

```
Loaded data from: X_train_raw.csv | Columns = 34 / 34 | Rows = 92945 -> 92945
Loaded data from: X_test_raw.csv | Columns = 33 / 33 | Rows = 4608 -> 4608
```

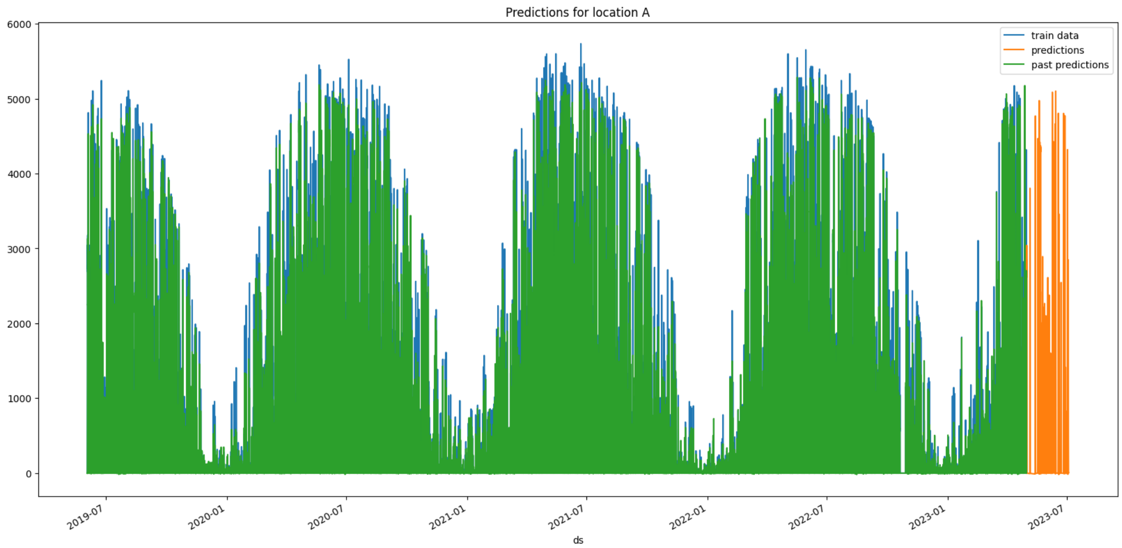
[16]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
 ↪"location"], left_on=["ds", "location"])

#test_data_merged
```

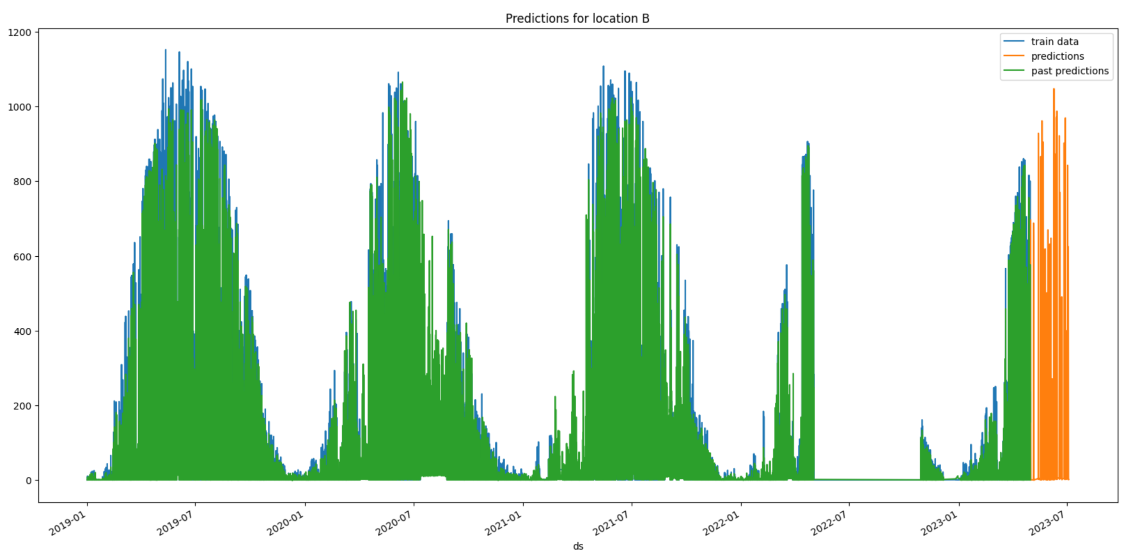
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

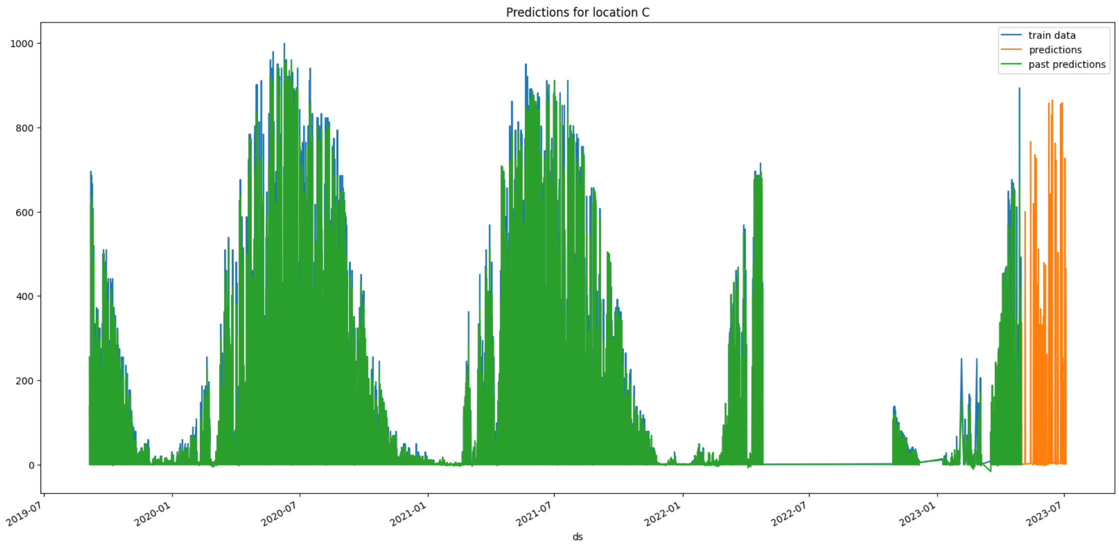
[17]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
```

```
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    past_pred = predictors[i].
↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,␣
↪"prediction"] = past_pred
```

[18]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")
```


Predictions for location A
train data
predictions
past predictions
6000
5000
4000
3000
2000
1000
0
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds


Predictions for location B
train data
predictions
past predictions
1200
1000
800
600
400
200
0
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds



[19]:
```python
# clip predictions smaller than 0 to 0
for pred in predictions:
    # print smallest prediction
    print("Smallest prediction:", pred["prediction"].min())
    pred.loc[pred["prediction"] < 0, "prediction"] = 0
    print("Smallest prediction after clipping:", pred["prediction"].min())

# concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

[19]:

	id	prediction
0	0	-1.060897
1	1	-0.598755
2	2	-1.056423
3	3	59.710995
4	4	311.332031
..	…	…
715	2155	68.854149
716	2156	39.890915
717	2157	9.410706
718	2158	4.226479
719	2159	1.412523

[2160 rows x 2 columns]

[20]:
```
# Save the submission DataFrame to submissions folder, create new name based on last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"), index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_97.csv
jall1a
```

[21]:
```
# save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

```
<IPython.core.display.Javascript object>
```

[22]:
```
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.ipynb"])
```

```
[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts…
  check_pandoc_version()
[NbConvertApp] Writing 135589 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 103636 bytes to notebook_pdfs/submission_97.pdf
```

[22]: CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output', 'notebook_pdfs/submission_97.pdf', 'autogluon_each_location.ipynb'], returncode=0)

[23]:
```
# feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,␣
↪time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using
4392 rows with 10 shuffle sets… Time limit: 600s…
        5441.74s        = Expected runtime (544.17s per shuffle set)
        322.9s  = Actual runtime (Completed 1 of 10 shuffle sets) (Early
stopping due to lack of time…)
```

[23]:

```
                                importance  stddev  p_value  n  p99_high  \
direct_rad_1h:J               1.821785e+02     NaN      NaN  1       NaN
clear_sky_energy_1h:J         9.408880e+01     NaN      NaN  1       NaN
clear_sky_rad:W               8.960991e+01     NaN      NaN  1       NaN
diffuse_rad_1h:J              8.077760e+01     NaN      NaN  1       NaN
direct_rad:W                  6.765221e+01     NaN      NaN  1       NaN
diffuse_rad:W                 6.723790e+01     NaN      NaN  1       NaN
sun_azimuth:d                 5.189177e+01     NaN      NaN  1       NaN
sun_elevation:d               3.894327e+01     NaN      NaN  1       NaN
effective_cloud_cover:p       2.830885e+01     NaN      NaN  1       NaN
total_cloud_cover:p           1.854761e+01     NaN      NaN  1       NaN
is_in_shadow:idx              1.564697e+01     NaN      NaN  1       NaN
cloud_base_agl:m              1.305668e+01     NaN      NaN  1       NaN
t_1000hPa:K                   1.262343e+01     NaN      NaN  1       NaN
ceiling_height_agl:m          1.259696e+01     NaN      NaN  1       NaN
relative_humidity_1000hPa:p   1.215808e+01     NaN      NaN  1       NaN
snow_water:kgm2               1.171797e+01     NaN      NaN  1       NaN
visibility:m                  1.116307e+01     NaN      NaN  1       NaN
wind_speed_10m:ms             1.031712e+01     NaN      NaN  1       NaN
pressure_100m:hPa             8.429827e+00     NaN      NaN  1       NaN
sfc_pressure:hPa              7.851625e+00     NaN      NaN  1       NaN
msl_pressure:hPa              7.671138e+00     NaN      NaN  1       NaN
fresh_snow_6h:cm              6.886095e+00     NaN      NaN  1       NaN
is_day:idx                    6.714718e+00     NaN      NaN  1       NaN
pressure_50m:hPa              6.671387e+00     NaN      NaN  1       NaN
precip_type_5min:idx          5.456353e+00     NaN      NaN  1       NaN
super_cooled_liquid_water:kgm2 4.694124e+00    NaN      NaN  1       NaN
fresh_snow_3h:cm              3.501178e+00     NaN      NaN  1       NaN
```

```
snow_depth:cm                   2.850314e+00     NaN      NaN  1       NaN
fresh_snow_1h:cm                2.693312e+00     NaN      NaN  1       NaN
is_estimated                   -1.552544e-08     NaN      NaN  1       NaN

                                p99_low
direct_rad_1h:J                     NaN
clear_sky_energy_1h:J               NaN
clear_sky_rad:W                     NaN
diffuse_rad_1h:J                    NaN
direct_rad:W                        NaN
diffuse_rad:W                       NaN
sun_azimuth:d                       NaN
sun_elevation:d                     NaN
effective_cloud_cover:p             NaN
total_cloud_cover:p                 NaN
is_in_shadow:idx                    NaN
cloud_base_agl:m                    NaN
t_1000hPa:K                         NaN
ceiling_height_agl:m                NaN
relative_humidity_1000hPa:p         NaN
snow_water:kgm2                     NaN
visibility:m                        NaN
wind_speed_10m:ms                   NaN
pressure_100m:hPa                   NaN
sfc_pressure:hPa                    NaN
msl_pressure:hPa                    NaN
fresh_snow_6h:cm                    NaN
is_day:idx                          NaN
pressure_50m:hPa                    NaN
precip_type_5min:idx                NaN
super_cooled_liquid_water:kgm2      NaN
fresh_snow_3h:cm                    NaN
snow_depth:cm                       NaN
fresh_snow_1h:cm                    NaN
is_estimated                        NaN
```

[24]:

```
# feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed,
 ↪time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using
5000 rows with 10 shuffle sets… Time limit: 600s…
        6395.23s        = Expected runtime (639.52s per shuffle set)
```

```
KeyboardInterrupt
```

```
[ ]: display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

#subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),␣
↪"autogluon_each_location.ipynb"])
```

```
[ ]: # import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])
```

```
# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream', 
 ↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```